Démarrer avec Ajax et le php: exemple d'application

Rédacteur: Alain Messin (Alain.Messin arobas obs-azur.fr)
CNRS UMS 2202
Admin06
24/09/2007

Le but de ce document est de permettre de démarrer dans le développement d'une application avec Ajax et php.

Je ne parlerais pas des avantages ou inconvénients, de la théorie ou de l'historique, mon but est seulement de pouvoir mettre le pied à l'étrier à ceux qui comme moi, sont persuadés que cette technique et quelques autres sont en train de changer la vision que l'on peut avoir des applications web et qui veulent s 'y investir. Le but est donc de décrire comment, le plus rapidement et simplement possible, faire tourner quelques exemples d'application Ajax, à partir desquels on pourra se faire sa propre expérience.

Pour autant, la partie consacrée à la préparation de l'application est assez longue, car je pense qu'on ne met pas de l' Ajax à toutes les sauces et qu'il est important de bien prendre en compte l'ensemble de l'ergonomie: c'est pourquoi je décris un exemple complet. L'exemple en question consiste à partir d'une application existante et d'y ajouter des outils Ajax afin de permettre une plus grande inter activité avec l'utilisateur, sans bouleverser l'application.

Je vais donc partir d'une application classique avec un menu à gauche de l'écran, qui charge différents « include » réalisant différentes fonctions. Je passe bien sûr sur toutes les problématiques de sécurité, de présentation etc.. lorsqu'elle ne sont pas directement liées à l'objet de cet article. On ajoutera progressivement à cette application les fonctionnalités permettant d'utiliser trois petits outils Ajax, qui fonctionneront donc indépendamment de l'application principale et sans rechargement des pages, puisque je le rappelle, Ajax est une technique qui permet l'interaction entre une page déjà chargée dans un navigateur et un serveur, sans rechargement de la page.

Pour cela, il faut disposer d'un navigateur (on testera avec Internet Explorer et la série Mozilla et autres), d'un serveur permettant l'exécution de php; il faudra activer le Javascript et/ou activeX.

On verra que pour créer une application Ajax, il faut mettre en place:

- des éléments de gestion de l'action (menus, fenêtres, boutons ...)
- un ou plusieurs élément(s) déclencheur(s)
- un script d'appel aux fonctions Ajax et de récupération des données du serveur
- une application serveur capable d'être appelée et de répondre au client
- un ou plusieurs containers destinés à mettre les données reçues

Je vais donc décrire l'application de départ « nue », et ajouter petit à petit les éléments permettant d'aboutir aux fonctionnement des outils Ajax voulus. J'ai essayé de décrire des outils variés qui peuvent employer différentes techniques de façon à se faire une idée déjà assez complète des possibilités. On verra que l'utilisation d' Ajax amène très vite à la manipulation du DOM (Document Object Model), mais le sujet est très vaste et quand même indépendant d' Ajax, donc je ne donnerais que quelques recettes de cuisine sur ce sujet.

1 L'application de départ:

On aura donc un fichier index.php, qui affichera les pages de l'application. Ce fichier sera en XHTML strict, et chargera une feuille de style style.css.

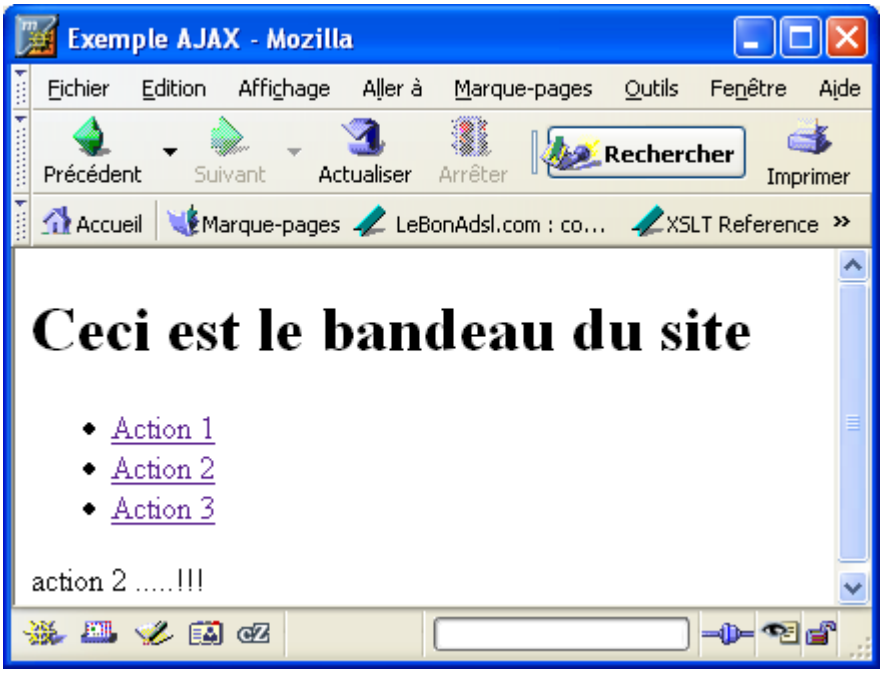
Pour les facilités de l'exemple, tous les fichiers seront placés dans le même dossier.

Le fichier index.php est donc le suivant:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="fr-FR" xml:lang="fr-FR">
 <head>
  <title>Exemple Ajax</title>

  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <link rel="stylesheet" type="text/css" href="style.css" media="screen" />
 </head>
 <body>
  <div class="bandeau">
   <h1>Ceci est le bandeau du site</h1>
  </div>
  <div class="menu">
   <ul>
    <li><a href="index.php?action=action1.php">Action 1</a>
    <li><a href="index.php?action=action2.php">Action 2</a>
    <li><a href="index.php?action=action3.php">Action 3</a>
   </ul>
  </div>
  <div class="action">
   <? include $_GET['action']; ?>
  </div>
 </body>
</html>
```

Ce script est en fait une page xhtml, telle que lors d'un clic sur un élément du menu, on rappelle cette page en incluant un fichier « action ». fonctions d'initialisation.

Sans le fichier style.css, ce script affichera:



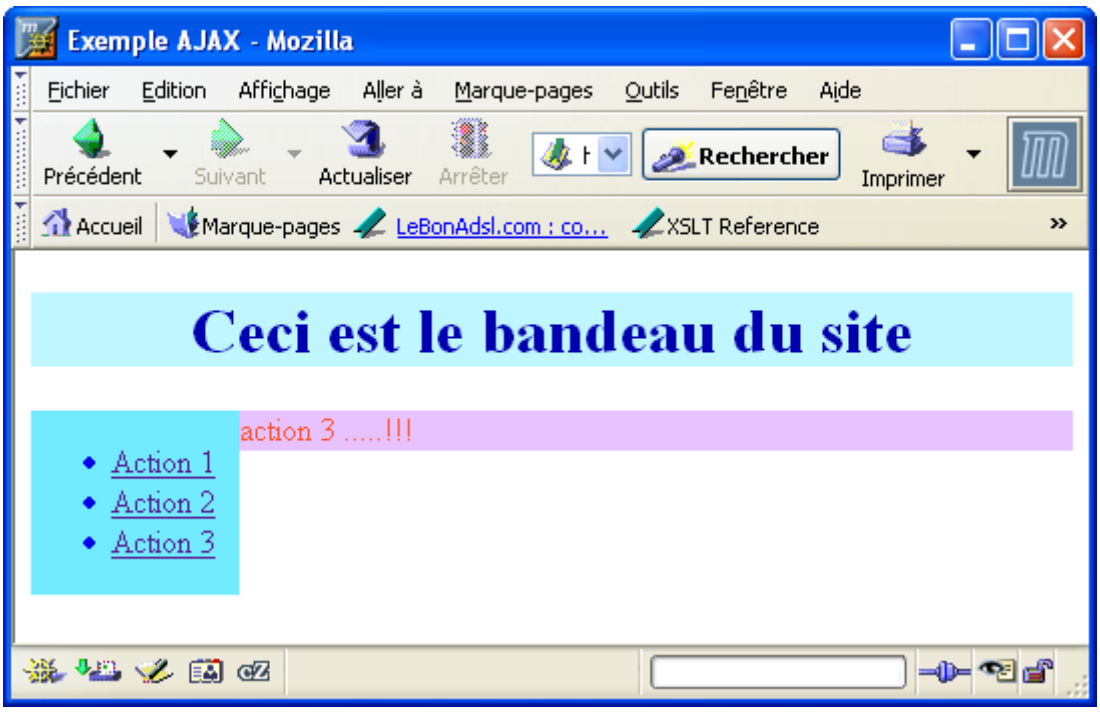
Si on a cliqué sur l'action 2.

Pour avoir un exemple plus parlant, on va ajouter des styles minimalistes (mais avec des couleurs bien distinctes !!!) dans le fichier style.css, de façon à présenter les différentes parties de la page de manière plus logique, le bandeau en haut, le menu à gauche et « l'action » au milieu.

Le fichier style.css sera celui-ci:

```
div {
 background-color: cyan;
}

.bandeau {
 width: 100%;
 background-color: #C0F6FF;
}

.bandeau h1 {
 color: #0000AE;
 text-align: center;
}

.menu {
 float: left;
 background-color: #72EAFF;
 color: #0000E9;
 width: 20%;
}

.action {
 background-color: #E7C2FF;
 color: #FF542E;
}
```

L'affichage deviendra alors:



2 Préparation à l'ajout des utilitaires Ajax

On veut donc ajouter des outils interactifs à notre application, sans la modifier outre mesure. Pour cela, on va ajouter dans le div « bandeau » un div « menuOutils », dans lequel on mettra un menu d'utilitaires, puis les « include » de nos outils.

```
    <div class="menuOutils">
     <span id="ouvreOutil1" class="ouvreOutil">outil1</span>
     <? include 'outil1.php'; ?>
     <span id="ouvreOutil2" class="ouvreOutil">outil2</span>
     <? include 'outil2.php'; ?>
     <span id="ouvreOutil3" class="ouvreOutil">outil3</span>
     <? include 'outil3.php'; ?>
    </div>
```

Remarquons que l'on a introduit des « id » dans les « span » d'affichage des menus. Nous aurons en effet besoin d'agir lorsque l'on cliquera sur ces faux boutons. On va également modifier le style bandeau h1 pour ne prendre que la moitié du bandeau (width: 50%;), pour laisser la place aux menus des outils. En cliquant sur le libellé de l'outil, on pourra l'ouvrir et l'utiliser; on va donc créer une classe « ouvreOutil ». L'outil comprendra un bouton de fermeture, nous créons donc une classe « fermeOutil ». Ces deux classes vont simuler des boutons sur lesquels on va cliquer, donc on définira « text-decoration: underline » et « cursor: pointer ».

```
.ouvreOutil {
 margin: 10%;
 background-color: #DFFFF4;
 text-decoration: underline;
 cursor: pointer;
}

.fermeOutil {
 position: relative;
 margin: 0;
 top:0;
 left:33em;
 width: 7em;
 height: 1,5em;
 font-size:1em;
 font-weight:bold;
 background-color: #DFFFF4;
 opacity:0.8;
 text-decoration: underline;
 cursor: pointer;
}
```

Il faudra aussi rendre ces faux boutons actifs, on le verra plus tard.

2.1 Les fenêtres des utilitaires:

Nous avons préparer les menus des outils, nous allons maintenant préparer les fenêtres contenant nos outils. Ici, j'ai choisi d'ouvrir des fenêtres identiques pour chaque outil, positionnées directement sous le menu outil et prenant sa largeur. Pour cela, les outils utiliseront une classe « outil », définie ci-dessous:

```
.outil {
 position: absolute;
 top: 100%;
 left: 0;
```

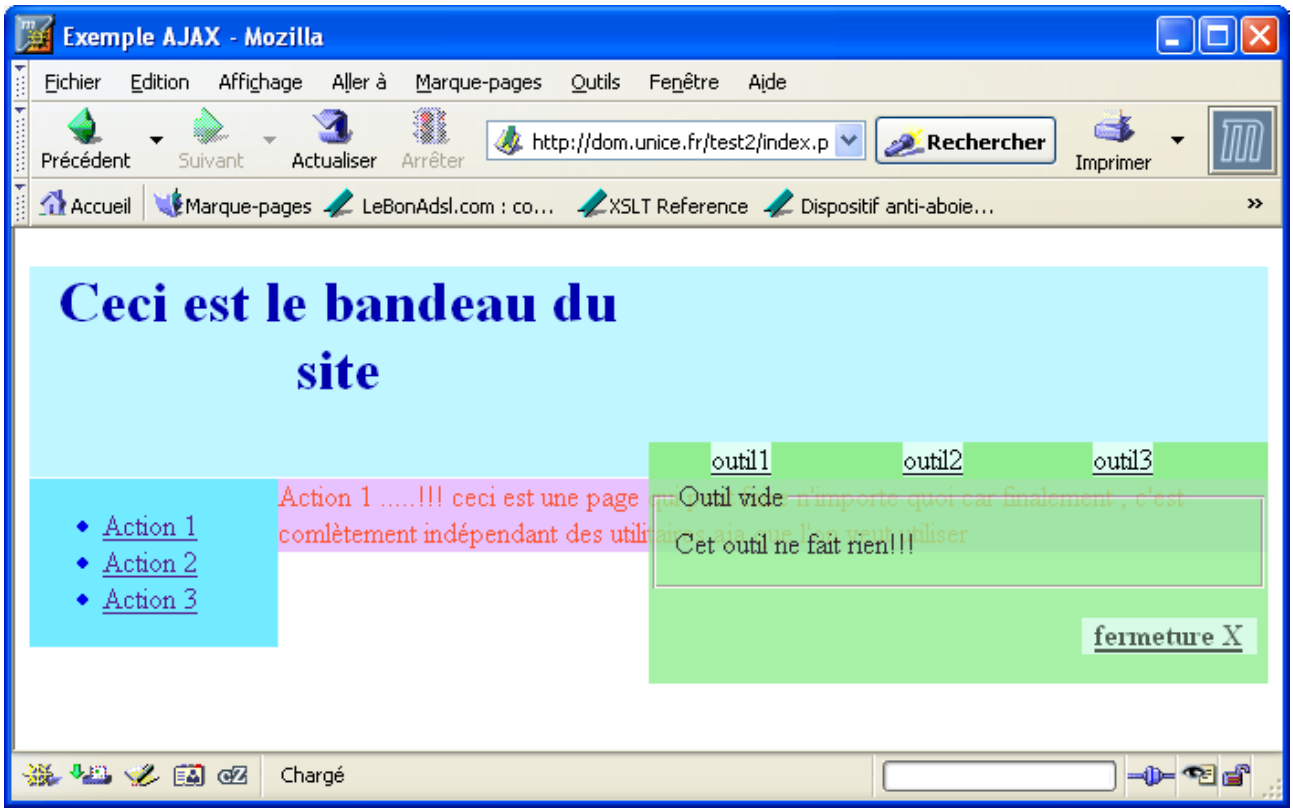
```
 width: 100%;
 padding:0;
 margin: 0;
 background-color:lightgreen;
 opacity: 0.8;
 display: none;
}
```

Le « display: none; » permet de cacher les outils au chargement de la page. On l'enlèvera pour la mise au point des fenêtres des outils. Opacity permet de rendre plus ou moins transparente la fenêtre, permettant pour les tests de voir ce qui est dessous (ça fonctionne différemment sous IE, et on le supprimera éventuellement).

Nous allons créer pour l'instant des outils « vides », simplement pour leur donner une structure et remarquer que l'on donne des « id » aux « div » containers et aux boutons de fermeture des outils, justement afin de pouvoir gérer l'affichage de chaque outil.

```
<?
// outil vide
echo '
<div id="outil1" class="outil">
 <fieldset>
  <legend>Outil vide</legend>
  <span>Cet outil ne fait rien!!!</span>
 </fieldset>
 <div id="fermeOutil1" class="fermeOutil">
  <p>fermeture X</p>
 </div>
</div>
';
?>
```

Notre site se présente maintenant ainsi:



2.2 Gestion des évènements

Il faut pouvoir faire apparaître et disparaître les outils, et il sera nécessaire pour déclencher nos actions Ajax de savoir gérer les évènements. Je vais donc insister un peu sur cette gestion d'évènement, bien que cela ne soit pas directement lié à Ajax; cela permettra également d'apprendre à gérer les évènements « proprement », c'est à dire indépendamment du contenu des pages, comme on le fait encore trop peu à l'heure actuelle. Tous les évènements et les actions Ajax seront réalisés par des scripts Javascript qu'il faut donc charger dans la page; on le fera d'une manière « propre », c'est à dire dans l'entête, et on les décomposera en modules:

- commun.js: comprendra toutes les fonctions générales et/ou communes à l'ensemble de l'application
- outil1, outil2, outil3.js comprendront respectivement les fonction liés aux fonctionnement de l'outil correspondant
- init.js comprendra les opérations d'initialisation de l'application.

On ajoutera donc après la référence à la feuille de style style.css

```
   <script type="text/javascript" src="commun.js"></script>
   <script type="text/javascript" src="outil1.js"></script>
   <script type="text/javascript" src="outil2.js"></script>
   <script type="text/javascript" src="outil3.js"></script>
   <script type="text/javascript" src="init.js"></script>
```

Tout d'abord, il faut pouvoir faire apparaître et disparaître les outils. Pour cela, on va donc éviter d'utiliser la syntaxe encore répandu « `onclick="fonction()` » positionné dans la balise du champ, pour utiliser des gestionnaires d'évènements qui vont prendre en charge les clics sur les boutons d'ouverture et de fermeture des outils. On aura donc l'enchaînement suivant:

- lors du chargement du script, un gestionnaire d'évènement (listener) est activé qui se déclenchera à la fin de chargement de la page (« onload »), provoquant ainsi l'appel de la fonction « init ».
- la fonction « init » permettra de créer les gestionnaires d'évènement qui seront déclenchés par les clics sur les boutons. Du fait que cette fonction « init » sera appelé à la fin du chargement de la page (par le déclenchement du listener « onload », on est sûr que tous les éléments concernés seront déjà chargés dans le DOM (Document Object Model) de la page.
- les gestionnaires appelleront les fonctions permettant l'ouverture et la fermeture des outils

On trouvera dans le fichier commun.js les fonctions addListener de création de gestionnaires d'évènements, trouveIdDeclencheur permettant de trouver l'ID de l'élément déclencheur de l'évènement, et la fonction existeChamp permettant de tester l'existence d'un champ, puis les fonctions spécifiques à notre application permettant d'ouvrir et de fermer les outils, en manipulant la propriété « display » des éléments « div » contenant les outils:

```
// ajoute listener
// compatible IE + mozilla & co
function addListener(element, action, fonction) {
    if (element.addEventListener)
        element.addEventListener(action, fonction, false);
    else if (element.attachEvent)
        element.attachEvent('on' + action, fonction);
} // addListener
function trouveIdDeclencheur(e) {
```

```
 // quel id a declenche l'evenement ?
 // compatible IE + mozilla & co
 if (window.event) {
  var id=window.event.srcElement.id;
 } else {
  var id=e.target.id;
 }
 return(id);
}

function existeChamp(id) {
 // retourne true si le champ d'id id existe, false sinon
 return (document.getElementById(id))? true: false;
}

function ofOutil(e) {
 //
 var id=trouveIdDeclencheur(e);
 if (id.substr(0,10)=='ouvreOutil') ouvreOutil(id);
 if (id.substr(0,10)=='fermeOutil') fermeOutil(id);
}

function ouvreOutil(id) {
 switch (id) {
  case 'ouvreOutil1':
   var idO='outil1';
   fermeOutil('outil2');
   fermeOutil('outil3');
   break;
  case 'ouvreOutil2':
   var idO='outil2';
   fermeOutil('outil1');
   fermeOutil('outil3');
   break;
  case 'ouvreOutil3':
   var idO='outil3';
   fermeOutil('outil1');
   fermeOutil('outil2');
   break;
 }
 if(existeChamp(idO)) document.getElementById(idO).style.display='inline';
}

function fermeOutil(id) {
 switch (id) {
  case 'fermeOutil1':
   var idF='outil1';
   break;
  case 'fermeOutil2':
   var idF='outil2';
   break;
  case 'fermeOutil3':
   var idF='outil3';
   break;
  default:
   var idF=id;
 }
 if(existeChamp(idF)) document.getElementById(idF).style.display='none';
}
```

Dans le fichier init.js, on trouvera la création des « listeners » nécessaires:

```
function init() {
 // les listeners lance la fonction ofOutil (ouvrefermeOutil)
 // listeners pour ecoute click ouverture outil
 addListener(document.getElementById('ouvreOutil1'),'click',ofOutil);
 addListener(document.getElementById('ouvreOutil2'),'click',ofOutil);
 addListener(document.getElementById('ouvreOutil3'),'click',ofOutil);
 // listeners pour ecoute click fermeture outil
 addListener(document.getElementById('fermeOutil1'),'click',ofOutil);
 addListener(document.getElementById('fermeOutil2'),'click',ofOutil);
 addListener(document.getElementById('fermeOutil3'),'click',ofOutil);
}

addListener(window,'load',init);
```

Détaillons le fonctionnement de cette partie:

- lors du chargement du script init.js (chargé en dernier, de façon à ce toutes les fonctions soient déjà définies), on crée un gestionnaire, qui lors de la fin de chargement de la page (il y a des querelles d'experts sur l'adéquation de cette fonction, mais passons...), va déclencher l'exécution de la fonction « init ». A noter qu'on fait dans la définition du gestionnaire une référence à la fonction et non un appel (d'où pas de parenthèses).
- la fonction « init » va à son tour définir les autres gestionnaires, avec l'avantage comme je l'ai souligné plus-haut, d'être sur à ce moment de l'existence des éléments utilisés. On fait références aux éléments de la page par leurs Id, en utilisant la fonction getElementById appliquée au document courant, et l'évènement « click » (et non pas onclick!) de la norme DOM. Tous les gestionnaires lanceront la fonction ofOutil lors d'un click sur l'élément voulu.
- la fonction addListener(element,action,fonction) permet de créer le gestionnaire de manière compatible aux navigateurs IE et compatibles DOM. Je passe sur le paramètre mis à false, qui gère le mode d'interception des évènements qui sera d'office le bouillonnement (bouillonnement ou Event bubbling en anglais, par rapport à la capture (Event capture), qui n'est d'ailleurs pas géré par IE, voir la littérature ou les spécifications DOM pour les curieux).
- à noter: lors du déclenchement d'un évènement, un objet « évènement » est passé en argument à la fonction de traitement (DOM) ou un objet windows.event est créé et accessible, ce qui permet à la fonction trouveIdDeclencheur(e) de trouveer l'Id de l'élément ayant déclenché l'action.
- les fonctions ofOutil, ouvreOutil et fermeOutil sont appelées selon l'ID déclenchant, pour ouvrir l'outil correspondant et fermer les autres, ou fermer un outil particulier.

2.3 Les outils

Il faut maintenant créer nos fichiers outil1.php, outil2.php, outil3.php, qui contiendront les outils Ajax que nous voulons pouvoir utiliser. Pour cela, j'ai opté pour des « include », comme pour les actions, ce qui permet facilement de développer ces modules sans (trop d') interaction avec le programme principal.

Comme exemple d'outils, j'ai choisi une mini-calculatrice, un bloc-notes et un outil de consultation de base de données, ce qui donnera un éventail assez divers déjà de ce qu'on peut faire avec Ajax. Ces outils seront donc disponibles dans n'importe quelle page de l'application, puisque indépendants des actions.

2.3.1 Outil1: calculette

L'outil 1 sera donc une mini-calculatrice sans touche, il suffira de taper une expression pour connaître le résultat. Évidemment, cela n'offre aucun intérêt si les calculs sont simples, on pourrait le faire directement en local en Javascript, mais si on doit faire appel à des ressources complexes, ou disponibles uniquement sur le serveur, l'approche Ajax peut se justifier.

```
<?
echo '
<div id="outil1" class="outil">
 <fieldset>
 <legend>Mini Calculatrice sans touche</legend>
 <span>Il suffit de taper une expression de calcul valide et de faire
return</span>
 <form id="outil1_form" action="" >
  <table>
   <tbody>
    <tr><td>Calcul à faire</td><td><input type="text" size="50" /></td></tr>
    <tr><td>Résultat</td><td><input type="text" size="50" disabled="disabled"
/></td></tr>
   </body>
  </table>
 </form>
</fieldset>
<div class="fermeOutil">
 <p id="fermeOutil1">fermeture X</p>
</div>
</div>
';
?>
```

On voit que cet outil comporte uniquement un formulaire avec le champ permettant de saisir l'expression à calculer, et un champ réceptacle du résultat renvoyé par le serveur. A noter que le formulaire ne définit pas d'action (l'attribut action n'est là que pour la conformité avec la norme XHTML strict) ni de méthode, car ce formulaire ne sera pas soumis, et n'existe que pour assurer la cohérence avec des champs de formulaire.

Le résultat du calcul sera mis dans un champ input désactivé, c'est souvent ce qui est utilisé car le plus simple à gérer, avec les fonctions Javascript déjà connus (document.getElementById(id).value= resultat).

2.3.2 Outil2: bloc-notes

L'outil 2 sera donc un bloc-notes permettant d'enregistrer ce que l'utilisateur note sans intervention de sa part, et de ré-utiliser ce contenu dans une autre page. Pour cela, on va simplement utiliser un « textarea ». On va aussi récupérer le contenu de la variable de session $_SESSION['bloc_notes'], qui sera initialisée dans le script php appelé par Ajax, ce qui permettra par exemple de garder le contenu du bloc_notes entre les différentes pages de l'application, voir de garder le contenu d'une session à l'autre, si l'utilisateur dispose d'un compte sur le serveur, etc...

Le reste est similaire à l'outil 1:

```
<?
$bloc_notes=(isset($_SESSION['bloc_notes']))? $_SESSION['bloc_notes']:'';
echo '
<div id="outil2" class="outil">
```

```
 <fieldset>
 <legend>Mini Bloc Notes</legend>
 <span>Mémorise automatiquement durant la durée de la session</span>
 <form id="outil2_form" action="">
  <table>
   <tr>
    <td><label>Texte</label></td>
    <td><textarea id="outil2_textarea" cols="20" rows="10"
>'.$bloc_notes.'</textarea></td></tr>
  </table>
 </form>
</fieldset>
<div class="fermeOutil">
 <p id="fermeOutil2">fermeture X</p>
</div>
</div>
';?>
```

2.3.3 Outil3: lecture d'une base de données

L'outil 3 permettra d'interroger une pseudo base de données, en affichant les caractéristiques et le nombre d'articles en stock d'un matériel informatique sélectionné dans une liste déroulante, liste qui sera elle-même initialisée par Ajax à partir de la base de données:

```
<?
echo '
<div id="outil3" class="outil">
 <fieldset>
 <legend>Consultation d\'une base de données</legend>
 <span>Acquiert les caractéristiques d\'un matériel informatique</span>
 <form id="outil3_form" action="">
  <table>
   <tr><td>Matériel</td><td>
    <select id="outil3_select">
      <option>Choisir un matériel</option>
      <option>Pas de choix disponible</option>
    </select></td></tr>
   <tr><td>CPU</td><td><input id="outil3_cpu" disabled="disabled" /></td></tr>
   <tr><td>Mémoire</td><td><input id="outil3_memoire" disabled="disabled"
/></td></tr>
   <tr><td>Disque Dur</td><td><input id="outil3_dd" disabled="disabled"
/></td></tr>
   <tr><td>Stock</td><td id="outil3_stock"></td></tr>
  </table>
 </form>
</fieldset>
<div class="fermeOutil">
 <p id="fermeOutil3">fermeture X</p>
</div>
</div>
';
?>
```

On voit que cet outil est similaire à l'outil 1, mais comporte un « select » permettant de choisir le matériel et quatre champs décrivant respectivement le CPU, la mémoire, le disque dur et le stock du matériel sélectionné, ce dernier étant affiché simplement dans une cellule de la table, afin d'illustrer une autre méthode de remplissage d'un container.

2.4 Ajout des déclencheurs

Avant d'envisager une action Ajax, il faut, comme indiqué au début, avoir la possibilité de déclencher une action. Le procédé dépend des outils:

- l'outil1 (calculette) doit déclencher son calcul lorsqu'on a terminé d'entrer la formule, lorsqu'on frappe un Return. On peut donc déclencher soit sur un Return (ce qui oblige à tester toutes les touches frappées, soit sur un changement du contenu du champ de saisie du calcul à faire. On va choisir ici de déclencher sur un Return, car la gestion de changement de contenu est très dépendante du navigateur sur les champs input.
- L'outil2 (bloc-notes) doit mémoriser le texte frappé à chaque instant: on peut procéder soit par test périodique du changement de valeur, soit par sauvegarde à chaque touche frappée. On choisira la première méthode, pour montrer une exemple de déclenchement d'Ajax par temporisation.
- L'outil3 (requête Bdd) doit déclencher une recherche lorsqu'on positionne le « select ». On va donc déclencher l'action sur le changement de valeur de ce champ. On initialisera le « select » au chargement de la page avec la liste des matériels.

On ajoutera dans le fichier init.js, dans la fonction init, les gestionnaires d'évènements correspondant à ce qu'on vient de définir, et les fonctions d'initialisation des outils:

```
// listeners pour le declenchement des actions
addListener(document.getElementById('outil1_saisie'),'keypress',lanceCalcul);
addListener(document.getElementById('outil3_select'),'change',lisBdd);
// recuperation du contenu initial du bloc-notes
recupBlocNotesDuServeur('outil2_textarea');
// initialisation de la sauvegarde du bloc-notes
setTimeout('sauveBlocNotesSurServeur("outil2_textarea")', 2000);
// initialisation de la liste du select
initSelect();
```

Bien sûr, on pourrait tout à fait inclure l'initialisation du bloc-notes et de la liste du « select » lors de l'affichage de la page principale de l'application, en utilisant du script PHP direct et non de l' Ajax; mais le but de cet exercice, je le rappelle, est aussi de rendre les outils aussi indépendants que possible de l'application principale.

On voit donc que lors de la saisie dans le champ id="outil1_saisie" (donc dans l'outil1, je rappelle que les id sont uniques), chaque appui sur une touche (« keypress ») déclenchera la fonction lanceCalcul, qui sera chargée de déclencher l'appel Ajax lorsqu'on appuiera sur Return. Le setTimeout permet d'initialiser toutes les deux secondes la sauvegarde du bloc-notes sur le serveur, tandis que chaque changement dans le « select » id="outil3_select" déclenchera la fonction lisBdd, qui se chargera de lire la base de données.

On créera les fonctions avec des alertes pour tester que les évènements sont bien pris en compte et que l'initialisation est correcte:

```
// dans outil1.js
function lanceCalcul(e) {
 // test si return, retourne true si oui false si non
 alert('touche outil1!');
}
// dans outil2.js
function  recupBlocNotesDuServeur(id) {
 alert ('recup '+id);
}
function sauveBlocNotesSurServeur(id) {
  alert('timeout outil2 sur '+id);
```

```
  setTimeout('sauveBlocNotesSurServeur("'+id+'")', 2000);
}
// dans outil3.js
function initSelect() {
 alert('init outil3!');
}
function lisBdd(e) {
 alert('changement outil3!');
}
```

Lorsqu'on s'est assuré que la mécanique de base ainsi construite fonctionne bien, on peut passer à la suite...

3 Fonctions et traitement Ajax:

Tout arrive! Maintenant que notre application est bien définie, nos outils prêts, et nos déclencheurs actifs, nous allons pouvoir enfin aborder vraiment Ajax. Pour cela, nous allons devoir, pour chaque outil, écrire les étapes suivantes:

- traitement et décodage du déclenchement
- préparation de la requête Ajax et envoi
- attente et réception du retour
- mis en place des résultats

Je vais le faire en détail pour l'outil 1, puis je décrirais uniquement les différences pour les autres outils.

Pour l'outil1, le déclenchement se fera pour chaque touche actionnée, mais il nous faut lancer le calcul uniquement lorsque c'est la touche Return qui est actionnée. On crée donc les fonctions lisCodeTouche(evt) et estReturn(evt) (evt étant l'évènement déclencheur) dans commun.js, pour reconnaître le Return (évidemment, le décodage est différent pour IE et les autres!):

```
function lisCodeTouche(evt) {
  if (window.event) {                          // si IE
   var CodeTouche = window.event.keyCode
  } else {                                     // si DOM
   var CodeTouche = evt.which;
  }
  return CodeTouche;
}

function estReturn(evt) {
 // test si return, retourne true si oui false si non
 var CodeTouche=lisCodeTouche(evt)
 return (CodeTouche == 13)? true : false;
}
```

3.1.1 Les requêtes Ajax

Outil 1: calculette

Une requête Ajax va consister à construire un requêteur, qui va gérer en fait le retour de l'information, de manière asynchrone avec l'application, et l'envoi la requête sous forme d'un « Post » (on pourrait aussi utiliser un « Get »), en appelant la page chargée du traitement de

l'information sur notre serveur, qui sera un script php capable d'acquérir les données transmises, de les traiter et de renvoyer l'information souhaitée.

On va tout d'abord préparer le « Post », en indiquant les variables transmises à la page appelée. Dans le cas de la calculette, c'est simple, on va transmettre l'expression saisie au clavier,en récupérant la valeur du champ « outil1_saisie » et en codant sa valeur sous forme compatible URI :

```
 if(!estReturn(e)) return false; // si pas return, on ne fait rien!!
 if(!existeChamp('outil1_saisie')) return false; // si pas de champ, non plus !
 var expression=document.getElementById('outil1_saisie').value;
 var data='expression='+encodeURIComponent(expression);          // a poster !!
```

On code la valeur récupérée dans l'élément de saisie par la fonction « encodeURIComponent » plutôt que de coder l'ensemble par « encodeURI », car cette dernière remplace le + par un espace, et ce n'est pas bon pour notre calcul!

Après avoir obtenu un requêteur « requester » (voir plus loin), on pourra expédier notre « Post » en précisant le type d'envoi ('post'), la destination (ici: le script 'calculette.php'), le mode asynchrone ('yes') qui signifie que l'on attend pas le retour pour continuer l'exécution, le type de données envoyées (Content-Type'), et les données en question (data):

```
requester.open('post','calculette.php',true);
requester.setRequestHeader('Content-Type',application/x-www-form-urlencoded');
requester.send(data);
```

La fonction d'obtention du requêteur « getRequester » que l'on ajoutera dans commun.js est indiquée ci-dessous, évidemment il faut traiter différemment IE et les navigateurs compatibles DOM, et même les différentes versions d'IE! Bref, sans entrer dans le détail, on crée un objet XMLHTTP qui est la base de tout le fonctionnement d' Ajax, vous pourrez aller fouiller la littérature pour en savoir plus!

```
function getRequester() {
 // obtient un requester Ajax compatible IE et DOM
 try {
  return new ActiveXObject('Msxml2.XMLHTTP');
 } catch(e) {
 }
 try {
  return new ActiveXObject('Microsoft.XMLHTTP');
 } catch (e) {
 }
 try {
  return new XMLHttpRequest();
 } catch (e) {
 }
 return false;
}
```

Dès le requêteur créé, on va définir la fonction à exécuter lorsque les données seront disponibles en provenance du serveur. Pour cela on utilise la propriété « onreadystatechange » du requêteur, cette fonction sera donc appelée à chaque changement d'état de la requête, mais nous n'agiront que si :

- la requête est terminée (readyState=4)
- l'état est OK (status=200)

Sinon on ignorera le changement d'état.

Là aussi, on pourra consulter la littérature pour affiner, si l'on veut, les actions à faire en cours de requête ou en cas d'échec de celle-ci. Mais lorsqu'on a mis au point la procédure, il n'y a

guère de raison d'échecs, sauf l'indisponibilité du serveur, et comme c'est en général celui qui permet de charger la page initiale, le risque est infime. On se contentera donc de ces tests dans notre fonctionnement habituel.

Ici, lors du retour des données depuis le serveur, on utilisera le retour « texte » du requêteur que l'on retrouvera dans la variable « result » et on le mettra tout simplement en place dans le container `'outil1_resultat'`.

```
var requester=getRequester();
requester.onreadystatechange=function(){
if (4==requester.readyState && 200 == requester.status) { // retour fait !
  var result=requester.responseText;
  if(existeChamp('outil1_resultat'))
             document.getElementById('outil1_resultat').value=result;
  else
             alert('outil1_resultat'+' existe pas!!! dans outil1');

  return true;
  }
};
```

On verra les autres outils que l'on peut utiliser aussi le résultat en format XML, ce qui est pratique lorsqu'on veut modifier la structure de la page (manipulation du DOM), ou qu'on ramène plusieurs résultats.

Pour notre outil 1, le fichier outil1.js sera donc en définitif celui ci-dessous, qui reprend les phases décrites ci-dessus, en ajoutant simplement après le « post » l'initialisation du champ container à : « En cours... », ce qui s'affichera en attente des résultats du serveur.

```
function lanceCalcul(e) {
 if(!estReturn(e)) return true;              //si pas return, on ne fait rien!!
 if(!existeChamp('outil1_saisie')) return false; //si pas de champ, non plus !
 var expression=document.getElementById('outil1_saisie').value;
 var data='expression='+encodeURIComponent(expression);         // a poster !!
 //calcul fait sur le serveur et retour par ajax
 var requester=getRequester();
 requester.onreadystatechange=function(){
  if (4==requester.readyState && 200 == requester.status) { // retour fait !
   var result=requester.responseText;
   if(existeChamp('outil1_resultat'))
document.getElementById('outil1_resultat').value=result; else
alert('outil1_resultat'+' existe pas!!! dans outil1');
   return true;
  }
 };
 // creation requester ajax et envoi des parametres
 requester.open('post','calculette.php',true);
 requester.setRequestHeader('Content-Type',
                                     'application/x-www-form-urlencoded');
 requester.send(data);
 // initialisation du champ de résultat
 if(existeChamp('outil1_resultat'))
  document.getElementById('outil1_resultat').value='En cours...';
 else alert('outil1_resultat'+' existe pas!!! dans outil1');
}
```

Outil 2: bloc_notes:

Dans la fonction init(), on fait appel à une fonction de récupération du contenu du bloc-notes, qui sera une fonction Ajax un peu particulière car on n'a pas de données à envoyer , mais juste une commande (recup):

```
function recupBlocNotesDuServeur(id) {
 var data='recup='; //attention, il faut le egal sinon variable $_POST pas cree

 //calcul fait sur le serveur arguments et retour par ajax
 var requester=getRequester();
 requester.onreadystatechange=function(){
  if (4==requester.readyState && 200 == requester.status) {
   var result=requester.responseXML;
   positionneResultats(result);
   return true;
  }
 };
 // creation requester ajax et envoi des parametres
 requester.open('post','bloc_notes.php',true);
 requester.setRequestHeader('Content-Type',
                                          'application/x-www-form-urlencoded');
 requester.send(data);
 return true;
}
```

On reconnaît les étapes expliquées pour l'outil 1, (mais pas de données à encoder), les seules différences étant l'appel au script bloc_notes.php et l'utilisation de requester.responseXML au lieu de requester.responseText, et la mise en place par une fonction « `positionneResultats(result)` », que l'on va regarder de plus près:

```
function positionneResultats(resultxml) {
 var data=resultxml.getElementsByTagName('contenuBlocNotes');
 if (!data[0]) return false;
 if (document.getElementById('outil2_textarea') && data[0].firstChild )
     document.getElementById('outil2_textarea').value=data[0].firstChild.data;
}
```

En fait le script PHP va renvoyer une chaîne de données au format XML, avec le contenu du bloc-notes déterminé par la balise « contenuBlocNotes », voir ci-après « les scripts PHP ».

On va donc extraire la liste des éléments de nom de balise « contenuBlocNotes », au moyen de la fonction « getElementsBytagName » appliquée à la structure resultxml retournée par notre script Ajax. Il n'y a qu'un élément de ce nom, on pointera donc sur cet élément (data[0]) et on mettra en place dans le container « outil2_textarea » le contenu (« data ») du premier enfant (noeud texte « firstChild » s'il existe. Evidemment, là c'est du DOM et il faut, ou connaître un peu, ou appliquer les recettes de cuisine pour les cas simples, c'est pourquoi j'ai utilisé ici cette méthode, pour varier un peu les plaisirs, car j'aurais aussi bien pu prendre la réponse texte et procéder comme pour la calculette.

Pour la mémorisation du bloc-notes, on a vu qu' on déclenche une fonction sauveBlocNotesSurServeur déclenchée par un setTimeout; on va utiliser cette fonction pour comparer le contenu du bloc-notes avec le contenu précédent, et appeler la fonction envoieBlocNotesAuServeur si ce contenu a changé (cette fonction Ajax sera un peu particulière dans le sens que l'on aura pas besoin du retour du serveur. Dans tous les cas cette fonction finira par un ré-armement du timer, pour se ré-appeler. On ajoutera ces fonctions au contenu de outil2.js, sachant que l'on ajoutera l'initialisation du vieux contenu dans init.js : var vContenu='':

```
function envoieBlocNotesAuServeur(contenu) {
 var data='sauve='+encodeURIComponent(contenu);

 var requester=getRequester();

 // on n'utilise pas le retour du serveur

 // envoi des parametres
 requester.open('post','bloc_notes.php',true);
 requester.setRequestHeader('Content-Type',
                                          'application/x-www-form-urlencoded');
 requester.send(data);
 return true;
}


function sauveBlocNotesSurServeur(id) {
  var nContenu=document.getElementById(id).value;
  if(nContenu!=vContenu) { // compare nouveau et vieux contenu
   vContenu=nContenu;      // mets à jour
   envoieBlocNotesAuServeur(nContenu);
  }
  setTimeout('sauveBlocNotesSurServeur("'+id+'")', 2000); // re-armement
}
```

Outil 3: lecture Base de données:

Dans cette application, on utilisera ce qu'on a vu dans les premiers outils; on a ajouté dans le fichier init.js la création du gestionnaire d'évènement déclenché sur le changement de valeur du « select » `'outil3_select'` , et l'appel à la fonction d'initialisation de ce même « select »:

Dans le fichier outil3.js, on trouvera donc:

- la fonction d'initialisation initSelect(), qui se chargera de gérer une requête Ajax qui permettra de constituer la liste des matériels à sélectionner, grâce à la fonction positionneSelect
- la fonction lisBdd, qui provoquera l'acquisition des caractéristiques du matériel choisi sur le serveur, et qui mettra en place les données reçues dans les containers grâce à la fonction positionneResultatsBdd.

Pour la fonction initSelect, le fonctionnement est identique à la fonction de récupération du bloc-notes, on pourrait d'ailleurs utiliser une fonction générique avec quelques paramètres, notamment le nom du script appellé, ici lis_bdd.php.

La fonction postionneSelect par contre est plus compliquée que les fonctions précédentes, car devant gérer de manière plus fine le DOM.

En effet, on va devoir commencer à retirer du « select » toutes les valeurs sélectionnables sauf la première (qui correspond à « Choisir un Matériel ») . Ces valeurs correspondent aux balises « option » du « select », on va donc enlever le « select.options[1] », tant que le nombre d'options du select, (c'est à dire sa longueur), sera supérieur à 1. Ceci sera réalisé par la fonction removeChild appliquée sur l'objet « select ».

Il faudra ensuite ajouter les valeurs fournies par le serveur dans la variable « result » sous forme d'une réponse XML. On va donc créer des éléments 'option', dont les valeurs de contenus et d'attributs seront fournies par la variable « data », qui récupérera dans le résultat XML la liste des éléments définis par la balise 'option' (« data=resultxml.getElementsByTagName('option') »).

L'attribut 'value' d'un élément 'i' sera donné par « data[i].getAttribute('value') », le libellé de l'option sera donné lui, par « data[i].firstChild.nodeValue », puisqu'étant le noeud texte fils du noeud 'option' de « data[i] ». On re-créera ensuite dans le DOM de la page pour chaque « data [i] » les noeuds « option » (« createTextNode »). On effectuera donc une boucle sur le nombre d'éléments de data, en créant les éléments et en leur affectant la bonne valeur et le bon attribut, le premier élément étant créé hors boucle, les suivants étant créés par clonage (« cloneNode »). Le menu déroulant du « select » sera ainsi alimenté avec toutes les références des matériels sélectionnables.

Pour la lecture de la base de données proprement dite, lors de la sélection d'un matériel dans le menu déroulant, la fonction lisBdd, après avoir vérifié l'existence du champ 'outil3_select', récupère l'index sélectionné (« selectedIndex »), puis la clef correspondante, à savoir la valeur de l'attribut 'value' (« childNodes[selectedIndex].getAttributes('value') »). cette valeur servira à fabriquer l'URI, pour le 'POST', l'appel Ajax étant identique aux autres et appelant le même script « lis_bdd.php » que la fonction précédente. La mise en place des données par la fonction « postionneResultatsBdd » est similaire à celle faite par la fonction correspondante de l'outil 2, sauf en ce qui concerne la mise en place du stock, pour lequel j'ai mis une autre variante, en plaçant directement la valeur dans le « td » de la table, et donc dans sa valeur « innerHTML ».

```
function initSelect(e) {
 var data=encodeURI('liste=');  //attention, il faut le= sinon variable pas
cree

 //calcul fait sur le serveur arguments et retour par ajax
 var requester=getRequester();
 requester.onreadystatechange=function(){
  if (4==requester.readyState && 200 == requester.status) {
   //var result=requester.responseText;
   var result=requester.responseXML;
   positionneSelect(result);
   return true;
  }
 };
 // creation requester ajax et envoi des parametres
 requester.open('post','lis_bdd.php',true);
 requester.setRequestHeader('Content-Type', 'application/x-www-form-
urlencoded');
 requester.send(data);
 return true;
}

function  positionneSelect(resultxml) {
 var data=resultxml.getElementsByTagName('option');
 //alert(data[0].firstChild.data);
 if (!existeChamp('outil3_select')) return false;
 var select=document.getElementById('outil3_select');
 while(select.length > 1){
  select.removeChild(select.options[1]);
 }
 var nouveau = document.createElement('option');
 for(var i = 0; i < data.length; i++){
  nouveau.setAttribute('value', data[i].getAttribute('value'));
  nouveau.appendChild(document.createTextNode(data[i].firstChild.nodeValue));
  select.appendChild(nouveau);
  nouveau = nouveau.cloneNode(false);
 }
 select.disabled = false;
}

function lisBdd(e) {
```

```
 if(!existeChamp('outil3_select')) return false;
 var selectedIndex=document.getElementById('outil3_select').selectedIndex;
 var
key=document.getElementById('outil3_select').childNodes[selectedIndex].getAttri
bute('value');
 var data=encodeURI('reference='+key);


 //calcul fait sur le serveur arguments et retour par ajax
 var requester=getRequester();
 requester.onreadystatechange=function(){
  if (4==requester.readyState && 200 == requester.status) {
   var result=requester.responseXML;
   positionneResultatsBdd(result);
   return true;
  }
 };
 // creation requester ajax et envoi des parametres
 requester.open('post','lis_bdd.php',true);
 requester.setRequestHeader('Content-Type', 'application/x-www-form-
urlencoded');
 requester.send(data);
 return true;
}

function positionneResultatsBdd(resultxml) {
 var data=resultxml.getElementsByTagName('cpu');
 if (document.getElementById('outil3_cpu') && data[0].firstChild)
     document.getElementById('outil3_cpu').value=data[0].firstChild.data;
 var data=resultxml.getElementsByTagName('mem');
 if (document.getElementById('outil3_memoire') && data[0].firstChild)
     document.getElementById('outil3_memoire').value=data[0].firstChild.data;
 var data=resultxml.getElementsByTagName('dd');
 if (document.getElementById('outil3_dd') && data[0].firstChild)
     document.getElementById('outil3_dd').value=data[0].firstChild.data;
 var data=resultxml.getElementsByTagName('stock');
 if (existeChamp('outil3_stock') && data[0].firstChild) {
  document.getElementById('outil3_stock').innerHTML=data[0].firstChild.data;
 }
}
```

4 Les scripts PHP:

Nous avons mis en place l'ensemble du dispositif coté client, il faut maintenant créer les scripts PHP qui sont appelés par le dispositif Ajax. La forme générale de ces scripts est très simple. On reçoit la requête (le « post » émis par Ajax), on exécute le traitement et on renvoie par un echo ou un print les données dans le format attendu par la requête Ajax, c'est à dire comme une simple chaîne de caractères ou comme un fragment XML.

Outil 1: calculette.php

Ce script, est on ne peut plus simple: il doit recevoir l'expression postée, l'évaluer, et renvoyer le résultat sous forme de chaîne. Cela tient en deux lignes (une fois encore, les questions de sécurité et autres ne sont pas évoquées, il faudrait ici éviter que le script puisse être appelé par n'importe qui, filtrer l'expression recueillie, éventuellement gérer une session utilisateur ...).

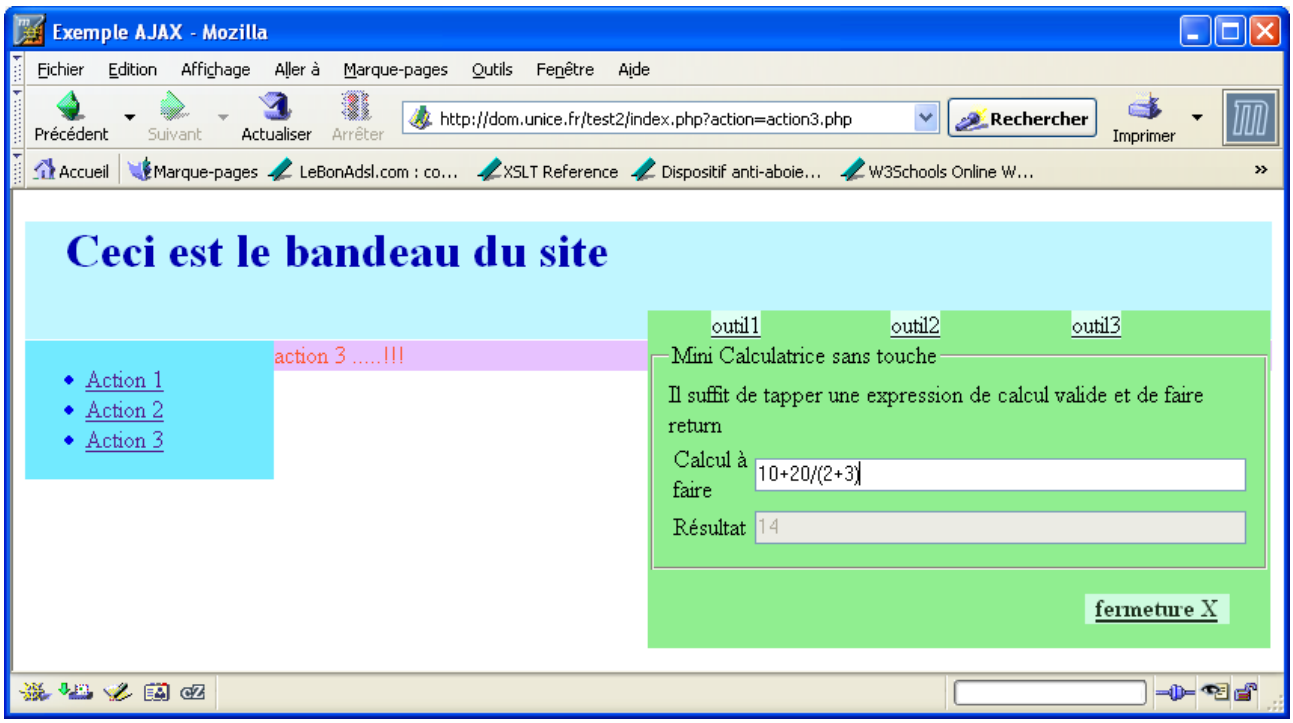
On peut utiliser la variable « $_POST », mais on peut avantageusement utiliser la variable $_REQUEST, qui recevra aussi bien les données des « POST » que des « GET » et qui nous

permettra ainsi, en forgeant une requête « GET » (plus facile à forger qu'une requête « POST »), d'appeler directement le script afin de vérifier sa sortie:

```
<?
$exp=$_POST['expression'];
if (eval('$v='.$exp.';')!==false) eval('echo '.$exp.';'); else echo 'Erreur !!!
';
?>
```

On a donc dans ce script simplement la lecture dans la variable $_POST de la valeur de l'expression, son évaluation par la fonction « eval » et l'affichage par un « echo » du résultat ou d'un message d'erreur. Il faut comprendre que ce script est appelé comme une page web et que ce qui est affiché par cette page est intercepté par le requêteur Ajax, et donc que tout ce qui est écrit par le script va être récupéré dans la variable « result » .C'est le cas aussi des messages d'erreurs éventuels, il faut donc s'assurer que le script PHP est bien mis au point, en utilisant par exemple un mode qui permet de l'appeler directement « hors Ajax » comme indiqué plus haut, ou l'écriture dans un fichier « log », afin de vérifier que le retour est bien ce qu'on attend.

L'outil1 complet fonctionne maintenant:

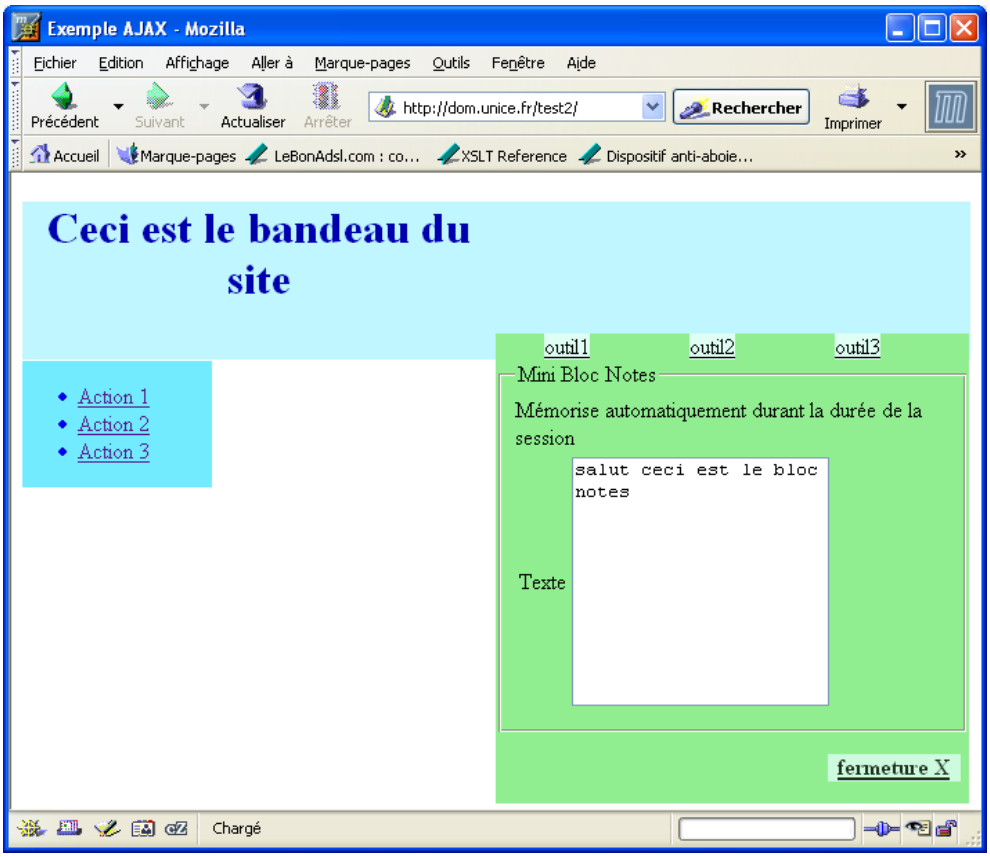


Outil 2: bloc_notes.php

Le script bloc_notes.php doit gérer deux commandes, la commande « recup » qui doit récupérer le contenu du bloc_notes sauvegardé dans la variable de session « $_SESSION['bloc_notes'] » et la fonction « sauve », qui fera l'inverse. Le script commencera donc par décoder la commande, et positionne une erreur si aucune des deux ne se trouve dans l'appel (on ne gérera toutefois pas cette erreur dans Ajax). Ensuite, le script codera les données éventuelles dans un fragment XML renvoyé au client par un « echo ». la génération du XML est faite simplement par création d'une entête de déclaration XML, et des données d'erreur et/ou de données encadrées par des balises « erreur » et « contenuBlocNotes », le tout renvoyé au client après l'envoi d'un « header » précisant que le contenu envoyé est du XML (voir ma note « exemple_xml_xslt_avec_php5 » pour plus d'informations sur la gestion du XML sous PHP5).

Le script complet bloc_notes.php, et la capture d'écran:

```
<?
session_start();
$erreur='Aucune';
$requete=false;
$data=false;

if(isset($_REQUEST['sauve'])) {
   $requete='sauve';
   $_SESSION['bloc_notes']=$_REQUEST['sauve'];
} elseif (isset($_REQUEST['recup'])) {
   $requete='recup';
   if(isset($_SESSION['bloc_notes'])) {
    $data=$_SESSION['bloc_notes'];
   }
} else {
  $erreur='Requete mal formee';
}

$str=utf8_encode("<?xml version='1.0' encoding='UTF-8' ?>\n");
$str.=utf8_encode("<resultats>\n");
$str.=utf8_encode('  <erreur>'.$erreur."</erreur>\n");
if ($erreur=='Aucune' and $requete=='recup') {
 $str.=utf8_encode('  <contenuBlocNotes>'.$data."</contenuBlocNotes>\n");
}
$str.=utf8_encode("</resultats>\n");
header('Content-Type: text/xml;charset=utf-8');
echo $str;
exit();
?>
```



Outil 3: lis_bdd.php:

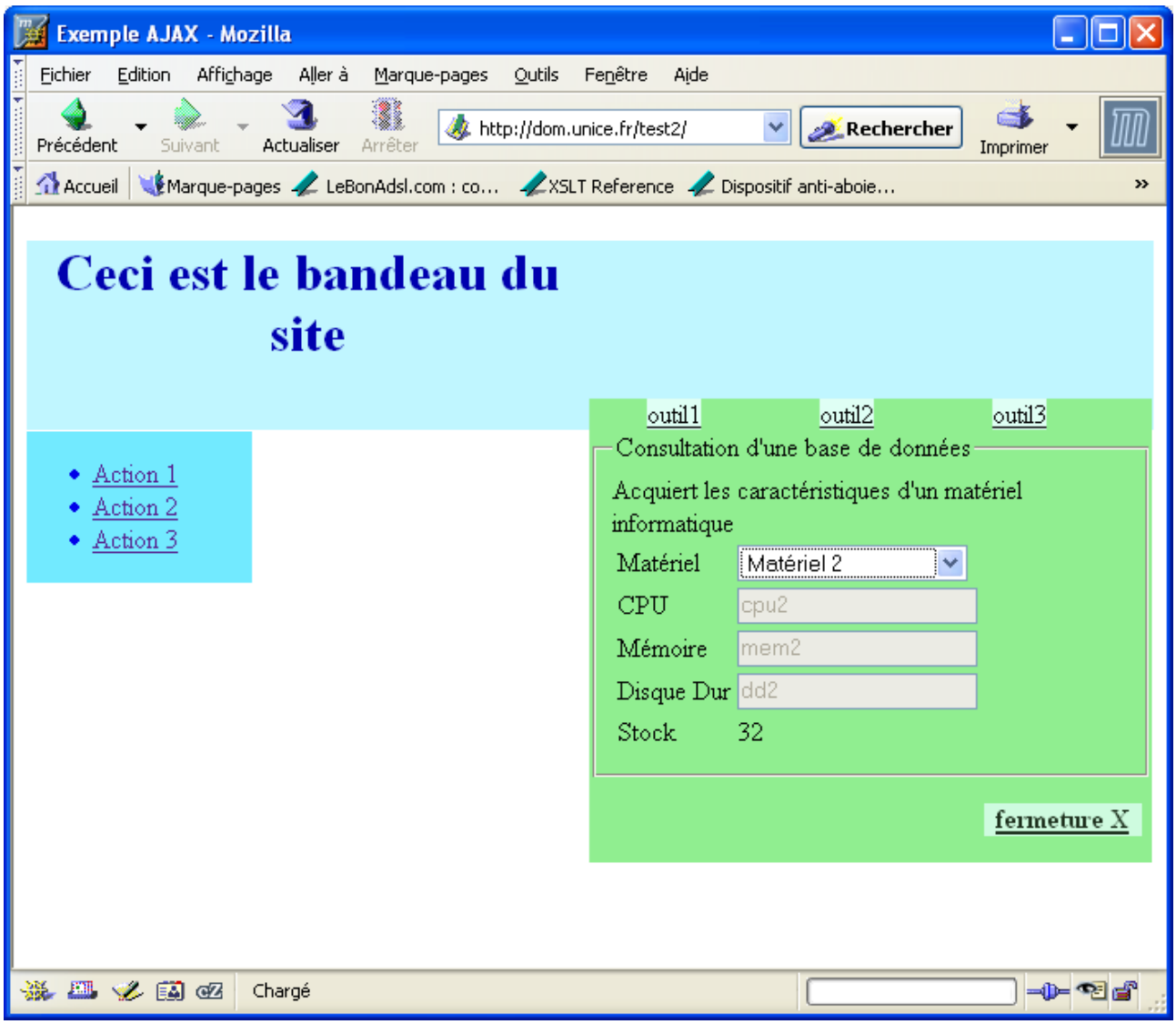
La aussi, le script doit gérer deux commendes différentes: « liste » et « reference », selon que l'on veut initialiser la liste du menu déroulant ou extraire une référence de la base. Cette base ne sera pas ici une vrai base, mais simplement un tableau comprenant les données nécessaires aux tests.

La première partie du script extrait les données voulues du tableau:

- dans la fonction « liste », on construit un fragment XML avec des balises « option » délimitant le libellé du matériel, tandis que l'attribut « value » est renseigné par la clef d'accès au matériel.
- dans la fonction « reference », on construit un fragment XML avec les balises « cpu », « mem », « dd », « stock ».
- quand les données sont fabriquées, le script les transmets au client de façon identique au script bloc_notes.php

Le script complet lis_bdd.php, et la capture d'écran:

```
<?
$materiels=array(
                 'm1'=>array('lib'=>'Matériel
1','cpu'=>'cpu1','mem'=>'mem1','dd'=>'dd1','stock'=>'5'),
                 'm2'=>array('lib'=>'Matériel
2','cpu'=>'cpu2','mem'=>'mem2','dd'=>'dd2','stock'=>'32'),
                 'm3'=>array('lib'=>'Matériel
3','cpu'=>'cpu3','mem'=>'mem3','dd'=>'dd3','stock'=>'7'),
                 'm4'=>array('lib'=>'Matériel
4','cpu'=>'cpu4','mem'=>'mem4','dd'=>'dd4','stock'=>'10'));
$erreur='Aucune';
$requete=false;
$data=false;
if(isset($_REQUEST['liste'])) {
   $requete='liste';
   $data='';
   foreach(array_keys($materiels) as $k) {
    $data.='<option value="'.$k.'">'.$materiels[$k]['lib']."</option>\n";
   }
} elseif (isset($_REQUEST['reference'])) {
   $requete='reference';
   $data=
       '<cpu>'.$materiels[$_REQUEST['reference']]['cpu']."</cpu>\n".
       '<mem>'.$materiels[$_REQUEST['reference']]['mem']."</mem>\n".
       '<dd>'.$materiels[$_REQUEST['reference']]['dd']."</dd>\n".
       '<stock>'.$materiels[$_REQUEST['reference']]['stock']."</stock>\n";
} else {
  $erreur='Requete mal formee';
}
$str=utf8_encode("<?xml version='1.0' encoding='UTF-8' ?>\n");
$str.=utf8_encode("<resultats>\n");
$str.=utf8_encode('  <erreur>'.$erreur."</erreur>\n");
if ($erreur=='Aucune') {
 $str.=utf8_encode($data);
}
$str.=utf8_encode("</resultats>\n");
header('Content-Type: text/xml;charset=utf-8');
echo $str;
exit();
?>
```


Exemple AJAX - Mozilla
Fichier Edition Affichage Aller à Marque-pages Outils Fenêtre Aide
Précédent Suivant Actualiser Arrêter
http://dom.unice.fr/test2/
Rechercher
Imprimer
Accueil Marque-pages LeBonAdsl.com : co... XSLT Reference Dispositif anti-aboie...
Ceci est le bandeau du site
outil1 outil2 outil3
Action 1
Action 2
Action 3
Consultation d'une base de données
Acquiert les caractéristiques d'un matériel informatique
Matériel Matériel 2
CPU cpu2
Mémoire mem2
Disque Dur dd2
Stock 32
fermeture X
Chargé

Documents complémentaires

Liens

Introduction aux techniques web
http://www.w3schools.com/

Javascript
http://www.commentcamarche.net/javascript/jsintro.php3

DOM
http://www.w3.org/TR/DOM-Level-2-Core/

DOM et gestion d'évènements
http://www.w3.org/TR/DOM-Level-2-Events/Overview.html#contents

XML avec PHP5
https://admin06.sophia.cnrs.fr/Documents%20partages/Techniques%20Web/070906version-exemple_xml_xslt_avec_php5.pdf

PHP:
http://www.php.net

Livres

Eyrolles Bien développer pour le Web 2.0 Christophe Porteneuve

Wyley Ajax for Dummies